CMPSC 465 Spring 2024 — Data Structures & Algorithms — Mehrdad Mahdavi and David Koslicki — HW 4

1. (20 pts.) Problem 1

(a) Similar to binary search, start by examining $A[\lfloor \frac{n}{2} \rfloor]$.

- If $A[\lfloor \frac{n}{2} \rfloor]$ is $-\lfloor \frac{n}{2} \rfloor$, then we have a satisfactory index.
- If $A[\lfloor \frac{n}{2} \rfloor] > -\lfloor \frac{n}{2} \rfloor$, then no element in the second half of the array can possibly satisfy the condition. To check this, note that all integers in the array are distinct, so each integer in the array is at least one greater than the previous element. Since $A[\lfloor \frac{n}{2} \rfloor]$ is already greater than $-\lfloor \frac{n}{2} \rfloor$, all the elements on its right are greater than their negative indices as well. Thus, the second half can be discarded in this case.
- if $A[\lfloor \frac{n}{2} \rfloor] < -\lfloor \frac{n}{2} \rfloor$, then by the same logic no element in the first half of the array can satisfy the condition.

We discard the half of the array that cannot hold an answer and repeat the same check until the satisfactory index has been found or all the elements in the array have been discarded.

(b) At each step we do a single comparison and discard at least half of the remaining array (or terminate), so the running time of this algorithm is given by the recurrence $T(n) = T(n/2) + O(1)$ and hence $T(n) = O(\log n)$ time by the master theorem. (Note we assume taking the negative takes constant time since we are not provided with bit information).

2. (20 pts.) Problem 2

(a)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^2 = \begin{bmatrix} a^2+bc & ab+bd \\ ca+dc & cb+d^2 \end{bmatrix} = \begin{bmatrix} a^2+bc & b(a+d) \\ c(a+d) & bc+d^2 \end{bmatrix}$$

Hence, the 5 multiplications $a^2, d^2, bc, b(a+d)$ and $c(a+d)$ suffice to compute the square.

(b) Two possible answers:

1) We have:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^2 = \begin{bmatrix} A^2+BC & AB+BD \\ CA+DC & CB+D^2 \end{bmatrix} \neq \begin{bmatrix} A^2+BC & B(A+D) \\ C(A+D) & BC+D^2 \end{bmatrix}.$$

Note that matrices don't commute! This means $BC \neq CB$ in general, so we cannot reuse that computation. Also, matrix multiplication only has left-distributivity and right-distributivity. As a result, $B(A+D) = BA+BD \neq AB+BD$ and $C(A+D) = CA+CD \neq CA+DC$. Thus, we end up getting more than 5 subproblems, and the recurrence $T(n) = 5T(n/2) + O(n^2)$ does not make sense.

2) Originally, our problem is to square a matrix. However, to compute $\begin{bmatrix} A & B \\ C & D \end{bmatrix}^2$, there is no way of making all the subproblems the same as our initial problem (squaring a matrix). So, this can't be claimed as a divide-and-conquer approach.

(c) Given two $n \times n$ matrices X and Y, create the $2n \times 2n$ matrix A:

$$A = \begin{bmatrix} 0 & X \\ Y & 0 \end{bmatrix}$$

Now, it suffices to compute A^2 to obtain the result of the matrix multiplication, as its upper left block will contain XY:

$$A^2 = \begin{bmatrix} XY & 0 \\ 0 & YX \end{bmatrix}$$

Let $S(n)$ be the time to square a $n \times n$ matrix. The general matrix multiplication XY can be calculated in time $S(2n)$. Given that $S(n) = O(n^c \log n)$, $S(2n) = O((2n)^c \log 2n) = O(2^c n^c \log 2n) = O(n^c \log n)$ since c is a constant as well as 2^c which can be ignored as a coefficient and $\log 2n = O(\log n)$.

Note: matrix A in the solution is not unique. The other possibilities for matrix A are $\begin{bmatrix} 0 & X \\ 0 & Y \end{bmatrix}$ ($A^2 = \begin{bmatrix} 0 & XY \\ 0 & Y^2 \end{bmatrix}$), $\begin{bmatrix} X & Y \\ 0 & 0 \end{bmatrix}$ ($A^2 = \begin{bmatrix} X^2 & XY \\ 0 & 0 \end{bmatrix}$), $\begin{bmatrix} Y & 0 \\ X & 0 \end{bmatrix}$ ($A^2 = \begin{bmatrix} Y^2 & 0 \\ XY & 0 \end{bmatrix}$), $\begin{bmatrix} 0 & 0 \\ Y & X \end{bmatrix}$ ($A^2 = \begin{bmatrix} 0 & 0 \\ XY & X^2 \end{bmatrix}$), and $\begin{bmatrix} 0 & Y \\ X & 0 \end{bmatrix}$ ($A^2 = \begin{bmatrix} YX & 0 \\ 0 & XY \end{bmatrix}$).

3. (20 pts.) Problem 3

(a) Suppose that x is the majority element in the original array and suppose first that n is even. Assume, by contradiction, that x is not the majority on either half of the array. If this is the case, the count of x should be $\leq n/4$ in the first half, and $\leq n/4$ in the second half. So overall, x appears at most $n/2$ times which implies that it is not the majority element and this leads to a contradiction. Similarly, if n is odd and x is not the majority in either half, the count of x should be $\leq (n-1)/4$ and $\leq (n+1)/4$ in each half, respectively. So, x appears at most $n/2$ times which is a contradiction.

(b) We scan the array counting how many entries equal x. If the count is more than $n/2$ we declare x to be a majority element. The algorithm scans the array and spends $O(1)$ time per element, so $O(n)$ time overall.

(c) We break the input array A into two halves, recursively call the algorithm on each half, and then test in A if either of the elements returned by the recursive calls is indeed a majority element of A. If that is the case we return the majority element, otherwise, we report that there is "no majority element". To argue the correctness, we see that if there is no majority element of A then the algorithm must return "no majority element" since no matter what the recursive call returns, we always check if an element is indeed a majority element. Otherwise, if there is a majority element x in A we are guaranteed that one of our recursive calls will identify x and the subsequent check will lead the algorithm to return x. Regarding time complexity, breaking the main problem into the two subproblems and testing the two candidate majority elements can be done in $O(n)$ time. Thus we get the following recurrence for the running time:

$$T(n) = \begin{cases} T(n) = 2T(n/2) + O(n), & \text{for } n>1 \\ O(1), & \text{for } n=1 \end{cases} \quad (1)$$

It follows from the master theorem that $T(n) = O(n \log n)$.

(d) After this procedure, there are at most $n/2$ elements left as at least one element in each pair is discarded. Now, let m be the majority element in the initial array. After we pair the elements, there will be pairs of fours types:

(i) We have pairs of the form $\{m,m\}$, where both elements of the pair are the majority element;

(ii) Pairs of the form $\{m,a\}$, where a is some element distinct from m;

(iii) Pairs of the form $\{a,a\}$ for some $a \neq m$; and

(iv) Pairs of the form $\{a,b\}$ for some a and b such that $a \neq m$, $b \neq m$ and $a \neq b$.

Let c_1, c_2, c_3 and c_4 be the number of pairs of each type, respectively. Let's assume first that n is even. Since pairs of the types $\{m,a\}$ and $\{a,b\}$ are discarded, it suffices for us to show that $c_1 > c_3$. Now, $2c_1 + c_2 > n/2$ and $c_1 + c_2 + c_3 + c_4 = n/2$. Combining these two facts, we get $c_1 + (n/2 - c_3 - c_4) > n/2$. This implies that $c_1 > c_3 + c_4$ and that $c_1 > c_3$ as desired. If n is odd, it again suffices to show that $c_1 > c_3$. Let $z = 1$ if the not-paired element is equal to m and 0 otherwise. Now, $c_1 + c_2 + c_3 + c_4 = (n-1)/2$ and $2c_1 + c_2 + z \geq (n+1)/2$. Then, $c_1 + z + ((n-1)/2 - c_3 - c_4) \geq (n+1)/2$ which implies that $c_1 + z \geq c_3 + c_4 + 1$. This implies that $c_1 + z > c_3 + c_4$. Now note that if $c_4 > 0$ or $z = 0$, then we have $c_1 > c_3$. Thus, the only case which remains is the case that $z = 1$ and $c_4 = 0$. In this case we may have $c_1 = c_3$. Remember that we defined A as initial array, and L as the elements left after pairing procedure. Note that so far, we showed if n is even then majority of A is same as majority of L, also if n is odd, and $c_4 > 0$ or $z = 0$, again the majority of A is same as L. Now, let's assume that n is odd, and $c_1 = c_3$, $c_4 = 0$ and $z = 1$. In this case L has c_1 number of m, and c_3 elements different than m, so the size of L would be $c_1 + c_3 = 2c_1$. Note that m would be a majority in L, if there are at least two elements different than m in L. Therefore, the only remaining case is when L has c_1 elements of same value like m, and c_1 elements of another element like a. In this case, we say we have a tie, and we use the unpaired element as the tie breaker. If we keep calling the pairing procedure on this array L, eventually L will be an empty array, and then we return the tie breaker as the majority.

(e) Let $\text{Maj}(A[0,\ldots,n-1], \text{tie-breaker})$ be the function that returns the majority element of A if there is one and if A is empty it returns the tie-breaker, and "no majority element" otherwise. Our implementation of $\text{Maj}(A[0,\ldots,n-1], \text{tie-breaker})$ will work as follows:

1. If $A.length == 1$, output $A[0]$. If $A.length == 0$, output tie-breaker.
2. Otherwise, pair up the elements of A arbitrarily, to get $n/2$ pairs. (If n is odd, update the value of tie-breaker with the unpaired element)
3. Look at each pair: if the two elements are different, discard both of them; if they are the same, keep just one of them. Let L be the resulting set of elements (as an array).
4. Let $m = \text{Maj}(L, \text{tie-breaker})$. If $m =$"no majority element", then there was no majority element in the recursive call, which can only mean that there was no majority to start with, so we output "error".
5. If $m \neq$"error", we use the method from part (b) to check that m is indeed the majority. If it is, we output m; if not, we output "no majority element".

The correctness of the algorithm follows from parts (b) and (d). The running time is given by the recurrence $T(n) = T(n/2) + O(n)$ which is $T(n) = O(n)$ by the master theorem.

4. (20 pts.) Problem 4

All recurrences can be solved by unrolling the recurrence and carefully arranging the terms, following the level-by-level approach from lectures (which is equivalent), or using various form of the Master Theorem. However, substitution or induction method is sometimes more convenient to showing the recurrence:

(a) In class, we saw that if $T(n) = a \cdot T(n/b) + O(n^d)$ for some $a > 0, b > 1$ and $d >= 0$, then:

$$T(n) = \begin{cases} O\left(n^d\right) & \text{if } d > \log_b a \\ O\left(n^d \log n\right) & \text{if } d = \log_b a \\ O\left(n^{\log_b a}\right) & \text{if } d < \log_b a \end{cases}$$

Using master Theorem ($a = 27, b = 3, d = 3$), we get $T(n) = O(n^3 \log n)$.

(b) $T(n) = 2T(n/7) + \sqrt{n} = O(\sqrt{n})$ by the Master theorem ($a = 2, b = 7, d = \frac{1}{2}$).

(c) $T(n) = T(n-1) + c^n = \sum_{i=1}^{n} c^i + T(0) = c(\frac{c^n - 1}{c-1}) + T(0) = O(c^n)$ (Formula for the partial sum of a geometric series starting from $i = 1$).

(d) $T(n) = T(n-1) + n^c = \sum_{i=1}^{n} i^c + T(0) = O(n^{c+1})$. (See homework 1 question 4c).

(e) $T(n) = 7T(n/4) + n = O(n^{\log_4 7})$ by the Master theorem.

(f) Unrolling, we have

$$T(n) = 2.25^n + 2.25^{n/2} + 2.25^{n/4} + 2.25^{n/8} + \dots$$

Then $T(n) \leq \sum_{i=0}^{n} 2.25^i = O(2.25^n)$. Hence, $T(n) = O(2.25^n)$.

(g) $T(n) = 49T(n/36) + n^{3/2} \log n = O(n^{3/2} \log n)$. By unrolling we have:

$$\begin{aligned}
\sum_{i=0}^{\log_{36} n} 49^i \left(\frac{n}{36^i}\right)^{3/2} \log\left(\frac{n}{36^i}\right) &= \sum_{i=0}^{\log_{36} n} \left(\frac{49}{36^{3/2}}\right)^i n^{3/2} \log\left(\frac{n}{36^i}\right) \\
&< \sum_{i=0}^{\log_{36} n} \left(\frac{49}{216}\right)^i O(n^{3/2} \log n) \\
&= O(n^{3/2} \log n) \sum_{i=0}^{\log_{36} n} \left(\frac{49}{216}\right)^i \\
&= O(n^{3/2} \log n) O(1) \\
&= O(n^{3/2} \log n)
\end{aligned} \tag{2}$$

Note that in the latter equation, we used the fact that the geometric series will be $O(1)$, since $\frac{49}{216} < 1$.

(h) By induction we will show that $T(n) = O(n)$. Assume for $k \leq n-1$, $T(k) \leq ck$, where $c \geq 12$. Then we have:

$$T(n) = T(\frac{n}{3}) + T(\frac{n}{6}) + T(\frac{n}{12}) + n \leq \frac{7}{12} cn + n \leq cn \tag{3}$$

As a result, we proved that $T(n) \leq 12n$, which means that $T(n) = O(n)$.

(i)
- First method (Unrolling).
  Let us consider the upper bound and suppose $T(n) \leq T(3n/7) + T(4n/7) + cn$ for a suitable constant $c > 0$. Note by unrolling that the size of each sub-problem at level k is of the form $S_{k,i} = \left(\frac{3}{7}\right)^i \left(\frac{4}{7}\right)^{k-i} n$ for some $i = 0, \dots, k$. Moreover, there are $\binom{k}{i}$ sub-problems of size $S_{k,i}$ at level k. The height of the recurrence tree is $\log_{7/4} n$. Hence, using the binomial expansion formula:

$$\begin{aligned}
T(n) &\leq \sum_{k=0}^{\log_{7/4} n} \sum_{i=0}^{k} \binom{k}{i} c \left(\frac{3}{7}\right)^i \left(\frac{4}{7}\right)^{k-i} n \\
&= cn \sum_{k=0}^{\log_{7/4} n} \sum_{i=0}^{k} \binom{k}{i} \left(\frac{3}{7}\right)^i \left(\frac{4}{7}\right)^{k-i} \\
&= cn \sum_{k=0}^{\log_{7/4} n} (\frac{3}{7} + \frac{4}{7})^k \\
&= cn \sum_{k=0}^{\log_{7/4} n} 1^k \\
&= O(n \log n)
\end{aligned} \tag{4}$$

- Second method (Induction).
  As the induction step, assume that for all $k < n$, we have $T(k) \le c_1 k \log k$. We want to show that there exists c_1 such that for all large enough n, $T(n) \le c_1 n \log n$. Since both $\frac{4n}{7}$, and $\frac{3n}{7}$ are less than n according to induction assumption we have: $T(\frac{4n}{7}) \le c_1 \frac{4n}{7} \log \frac{4n}{7}$, and $T(\frac{3n}{7}) \le c_1 \frac{3n}{7} \log \frac{3n}{7}$. Now using recurrence relation for $T(n)$ we can write:

$$\begin{aligned} T(n) \le T(\frac{3n}{7}) + T(\frac{4n}{7}) + cn &\le c_1 \frac{3n}{7} \log \frac{3n}{7} + c_1 \frac{4n}{7} \log \frac{4n}{7} + cn \\ &= c_1 \frac{3n}{7} \log \frac{3}{7} + c_1 \frac{3n}{7} \log n + c_1 \frac{4n}{7} \log \frac{4}{7} + c_1 \frac{4n}{7} \log n + cn \\ &= c_1 n \log n + c_1 n (\frac{3}{7} \log \frac{3}{7} + \frac{4}{7} \log \frac{4}{7}) + cn \\ &= c_1 n \log n + cn - c_1 n (\frac{3}{7} \log \frac{7}{3} + \frac{4}{7} \log \frac{7}{4}) \end{aligned} \tag{5}$$

  Now, note that if we let $c_1 = \frac{c}{\frac{3}{7} \log \frac{7}{3} + \frac{4}{7} \log \frac{7}{4}}$, and replace it in the latter equation, we get $cn - c_1 n (\frac{3}{7} \log \frac{7}{3} + \frac{4}{7} \log \frac{7}{4}) = 0$, which means that $T(n) \le c_1 n \log n$. So, induction step is complete, and we proved that $T(n) = O(n \log n)$

(j) Note by unfolding that

$$T(n) = n^{\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^k}} T(n^{\frac{1}{2^k}}) + 11kn.$$

(This can also be proved, for example, by induction.) Now if we let $k = \log_2 \log_2 n$, we have $n^{\frac{1}{2^k}} = 2$ which is constant. Since $\frac{1}{2} \le \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^k} \le 1$, we have $T(n) = \Theta(n \log \log n)$.

5. (20 pts.) Problem 5

The intuition for the divide-and-conquer solution is to split the vector v into a top half v_T with the first $\lfloor n/2 \rfloor$ elements and a bottom half v_B with the rest of the elements. We'll similarly split the result of the multiplication $(H_k v)_T$ and $(H_k v)_B$. Using this notation and the recursive nature of H_k, we can split $H_k v$ as follows:

$$H_k v = \begin{bmatrix} (H_k v)_T \\ (H_k v)_B \end{bmatrix} = \begin{bmatrix} H_{k-1} v_T + H_{k-1} v_B \\ -H_{k-1} v_T - H_{k-1} v_B \end{bmatrix} = \begin{bmatrix} H_{k-1}(v_T + v_B) \\ H_{k-1}(-v_T - v_B) \end{bmatrix}$$

This shows that we can find $H_k v$ by calculating $v_T + v_B$ and $-v_T - v_B$ and using this algorithm to recursively multiply $H_{k-1}(v_T + v_B)$ and $H_{k-1}(-v_T - v_B)$. The running time of this algorithm is given by the recurrence relation $T(n) = 2T(\frac{n}{2}) + O(n)$, where the linear term is the time taken to perform the two sums. This has solution $T(n) = O(n \log n)$ by the Master theorem. Note that it can be solved in linear time if you notice that one matrix-vector product is the negative of the other.

This question can also be solved in linear time by summing the elements of the vector together in linear time, then placing the resulting sum and its negative in the appropriate locations in the output vector. That is, the negative of the summation should go to the indexes where the corresponding row of H_k is negative.

6. (0 pts.) Acknowledgments

(a) I did not work in a group.
(b) I did not consult with anyone other than my group members.
(c) I did not consult any non-class materials.

Rubric:

Problem 1, 20 pts

(a) 15 points
6 points: provide an algorithm that achieves what we want
4 points: it's truly faster than the brute force algorithm
5 points: explain why it's correct

(b) 5 points
3 points: provide a running time analysis that makes sense
2 points: reach to a correct conclusion on the running time
Note:

- Some students might analyze the running time correctly, but the given algorithm is not faster than the brute force way. In this case, they'll get full points for part (b) but lose 4 points for part (a).
- Some other students might provide an algorithm that's indeed faster than the brute force approach, but their running time analysis has certain issues. In this case, they'll have the 4 points for part (a) second rubric but lose points for part (b).

Problem 2, 20 pts

(a) 5 points
3 points: correctly expand the squaring of a 2×2 matrix
2 points: correctly identify the 5 needed multiplications

(b) 6 points as long as the student provides either of the possible answers

1) 3 points: correctly expand the squaring of a matrix with 4 sub-matrices and point out the non-commutativity property
   3 points: explain how the non-commutativity property introduces an issue for the statement
2) 3 points: point out the sub-problems of squaring a matrix are not necessarily squaring a matrix
   3 points: point out it's no longer divide-and-conquer if the main problem and its sub-problems are different

(c) 9 points
3 points: the explanation of squaring a $2n \times 2n$ matrix can also be done in $O(n^c \log n)$ time makes sense
2 points: specify a reasonable $2n \times 2n$ matrix A
4 points: explain how they can obtain the result of a general matrix multiplication from A^2

Problem 3, 20 pts

(a) 6 pts for right proof

(b) 5 pts for appropriate explanation for running time $O(n)$

(c) 5 pts for right algorithm and right recurrence

(d) 2 pts for right proof

(e) 2 pts for right algorithm description

Problem 4, 20 pts

There are total of 10 parts, each part is worth 2 pts. For the parts that can be solve by master theorem, only correct answer get the full points, and there is not partial credit for these parts. For, the parts that master theorem can not be applied, unrolling correctly is worth 1 pts, and showing the final answer is worth 1 pts.

Problem 5, 20 pts

Divide-and-conquer approach

1. 5 pts for identifying that the problem can be split into top and bottom
2. 10 pts for splitting the multiplication into 2 multiplications of size $n/2$
3. 5 pts for showing the running time is $O(n \log n)$; $O(n)$ is also acceptable

Matrix-vector multiplication approach

1. 15 pts for correctly indicating where the vector summation should go in the output vector
2. 5 pts for showing the running time is $O(n \log n)$; $O(n)$ is also acceptable